ГЛАВА 1

ПУТЬ РАЗРАБОТКИ

Цель этой книги — научить вас мыслить как настоящий программист. Этот способ сочетает в себе особенности мышления математика, инженера и ученого. Как математики компьютерные специалисты используют формальные языки для выражения идей (в частности, вычислений). Как инженеры они что-то проектируют, собирают отдельные компоненты в системы и оценивают компромиссы между альтернативами. Как ученые они наблюдают за поведением сложных систем, формируют гипотезы и тестируют прогнозы.

Единственный самый важный навык для разработчика — умение **находить решение задачи**. Для этого он должен сформулировать задачу, подойти творчески к поиску решения, а затем точно и ясно его реализовать. Как видите, обучение программированию — это прекрасная возможность попрактиковаться в решении задач. Вот почему эта глава называется «Путь разработки».

С одной стороны, вы будете учиться программировать, что само по себе полезный навык. С другой — вы будете использовать программирование как средство для достижения цели. По мере того как мы будем продвигаться дальше, вы поймете, о чем я.

ЧТО ТАКОЕ ПРОГРАММА?

Программа — это последовательность инструкций, в которых указано, как выполнять вычисления. Вычисления могут быть математическими, такими как решение системы уравнений или поиск корней многочлена, но это также могут быть символические вычисления, например поиск и замена текста в документе, или что-то графическое, например обработка изображения или воспроизведение видеоролика.

Детали реализации выглядят по-разному на разных языках, но несколько основных инструкций универсальны для любого языка:

— *ввод данных (input):*
  Получение данных с клавиатуры, из файла, по сети или с другого устройства.
— *вывод данных (output):*
  Отображение данных на экране, сохранение их в файл, отправка по сети и так далее.
— *математические операции (math):*
  Выполнение основных математических операций, таких как сложение и умножение.
— *условное выполнение (conditional execution):*
  Проверка определенных условий и выполнение соответствующего кода.
— *повторение (repetition):*
  Выполнение некоторого действия несколько раз, часто с некоторыми изменениями.

Верьте или нет, но это все, что нужно знать. Каждая программа, которую вы когда-либо использовали, независимо от ее сложности, состоит из таких инструкций. Таким образом, вы можете представить программирование как процесс разбиения большой и сложной задачи на всё более мелкие подзадачи, пока подзадачи не станут достаточно простыми, чтобы их можно было сформулировать с помощью одной из этих инструкций.

ЗАПУСК PYTHON

Работа с Python начинается с установки Python и связанного программного обеспечения на компьютер. Если вы знакомы с вашей операционной системой и особенно если вы знакомы с интерфейсом командной строки, у вас не должно возникнуть проблем. Но новичкам сложновато изучать системное администрирование и программирование одновременно.

Чтобы облегчить задачу, я рекомендую запустить Python в браузере. Позже, когда вы освоитесь, я предложу вам установить Python на компьютер.

Существует несколько веб-сайтов для запуска Python. Если у вас уже есть любимый, можете смело использовать его. В противном случае я рекомендую

PythonAnywhere. Подробные инструкции по началу работы приведены на странице **http://tinyurl.com/thinkpython2e**.

Существует две версии языка, Python 2 и Python 3. Они очень похожи, поэтому, если вы изучите одну из них, то легко сможете использовать и другую. На самом деле есть только несколько отличий, с которыми вы столкнетесь как новичок. Эта книга написана под Python 3, но я добавил несколько примечаний, касающихся и Python 2.

Интерпретатор (interpreter) Python — это программа, которая анализирует, обрабатывает и выполняет код Python. В зависимости от установленной операционной системы вы можете запустить интерпретатор, щелкнув мышью по значку или набрав слово python в командной строке. В случае успешного запуска вы должны увидеть примерно следующий результат:

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Первые три строки содержат информацию об интерпретаторе и операционной системе, в которой он запущен, поэтому у вас сведения могут отличаться. Но вы должны проверить, что номер версии (в этом примере — `3.4.0`) начинается с 3, что указывает на то, что вы используете Python 3. Если номер версии начинается с 2, вы работаете (как вы уже догадались) с Python 2.

Последняя строка представляет собой **приглашение (prompt)**, которое указывает, что интерпретатор ожидает ввод команды от пользователя. Если вы наберете в строке 1 + 1 и нажмете клавишу **Enter**, интерпретатор отобразит результат:

```
>>> 1 + 1
2
```

Теперь вы готовы начать учиться. С этого момента я предполагаю, что вы знаете, как использовать интерпретатор Python.

ПЕРВАЯ ПРОГРАММА

Традиционно первая программа, которую пишут на любом новом языке программирования называется Hello, World! Все, что она делает, это отображает слова Hello, World! (то есть «Привет, мир!»). На языке Python программа выглядит так:

```
>>> print('Привет, мир!')
```

Это пример **инструкции печати**, хотя на самом деле она ничего не печатает на бумаге. Она отображает результат на экране. В этом случае результатом будут следующие слова:

```
Привет, мир!
```

Кавычки в программе отмечают начало и конец отображаемого текста; они не видны в выводе.

Скобки `()` указывают, что `print` — это функция. Мы рассмотрим функции в главе 3.

В Python 2 инструкция печати немного отличается; это не функция, поэтому скобки не используются.

```
>>> print 'Привет, мир!'
```

Это различие будет иметь смысл далее, но для начала достаточно просто об этом помнить.

АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ

После простенькой программы Hello, World наш следующий шаг — арифметика. В языке Python есть **операторы (operators)**, которые выглядят как специальные символы, представляющие вычисления, такие как сложение и умножение.

Операторы `+`, `-` и `*` выполняют сложение, вычитание и умножение, соответственно, как показано в следующих примерах:

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
```

Оператор `/` выполняет деление:

```
>>> 84 / 2
42.0
```

Вы можете спросить, почему результат `42.0`, а не `42`. Я объясню это в следующем разделе.

Наконец, оператор ** выполняет возведение в степень, то есть умножает число на это же число указанное количество раз:

```
>>> 6 ** 2 + 6
42
```

В некоторых других языках для возведения в степень используется символ ^, но в Python это побитовый оператор, называемый XOR (исключающее «ИЛИ»). Если вы не знакомы с побитовыми операторами, результат вас удивит:

```
>>> 6 ^ 2
4
```

Я не буду рассматривать побитовые операторы в этой книге, но вы можете прочитать о них по адресу **http://wiki.python.org/moin/BitwiseOperators**.

ЗНАЧЕНИЯ И ТИПЫ

Значение (value) — это одно из основных понятий, с которым работает программа, например буква или цифра. Мы уже видели некоторые значения: `2`, `42.0` и `'Привет, мир!'`

Эти значения принадлежат разному **типу**: `2` — это **целое число (`int`)**, `42.0` — **число с плавающей точкой*** **(floating-point number)**, а `'Привет, мир!'` — **строка (`str`)**.

Если вы не знаете, какого типа указанное значение, интерпретатор может подсказать вам:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Привет, мир!')
<class 'str'>
```

Слово `class` здесь используется для обозначения типа.

Неудивительно, что целые числа принадлежат к целочисленному типу `int`, строки относятся к `str`, а числа с плавающей точкой — это `float`.

---

* Числа с плавающей точкой — вещественные числа, такие как 1.45, 0.00453 или –3.789. В России также может использоваться термин «плавающая запятая», но так как в большинстве языков программирования для отделения дробной части от целой используется именно точка, далее будет использовано международное обозначение. *Прим. перев.*

А как насчет таких значений, как `'2'` и `'42.0'`? Они выглядят как числа, но указаны в кавычках, как строки:

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

Интерпретатор определяет их как строки.

При вводе большого целого числа иногда хочется использовать запятые как разделители между группами цифр, например так: `1,000,000`. В этом случае Python не опознает *целое число*:

```
>>> 1,000,000
(1, 0, 0)
```

Это совсем не то, что мы ожидали! Python интерпретирует `1,000,000` как последовательность целых чисел через запятую. Мы узнаем больше об этом виде последовательности позже.

ФОРМАЛЬНЫЕ И ЕСТЕСТВЕННЫЕ ЯЗЫКИ

На **естественных языках** говорят люди; это, например, английский, испанский и французский языки. Они не были спроектированы людьми (хотя люди и пытаются соблюдать в них некий порядок); они развивались естественно.

Формальные языки разрабатываются людьми для определенных целей. Например, математические символы представляют собой формальный язык, который особенно хорош для обозначения отношений между числами и символами. Химики используют формальный язык для представления химической структуры молекул. И самое важное:

> **Языки программирования — это формальные языки, предназначенные для выражения вычислений.**

Формальные языки, как правило, имеют строгие **синтаксические** правила, которым подчиняется структура кода. Например, в математике утверждение 3 + 3 = 6 имеет правильный синтаксис, а 3 + = 3$6 — нет. В химии H_2O — синтаксически правильная формула, а *2Zz* — нет.

Синтаксические правила бывают двух видов: относящиеся к **токенам** и к структуре. Токены — основные элементы языка, такие как слова, числа

и химические элементы. Одна из проблем с 3 + = 3$6 заключается в том, что это недопустимый в математике токен (по крайней мере, по моим сведениям). Аналогично, *2Zz* недопустимо, так как нет элемента с сокращением *Zz*.

Второй тип правил синтаксиса относится к способу объединения токенов. Операция 3 + = 3 недопустима, поскольку хотя символы + и = и являются допустимыми токенами, их нельзя использовать один сразу за другим. Точно так же в химической формуле нижний индекс указывается после имени элемента, а не перед.

Это х0рошо структуриров@нное пр3длож3ние с недопустимым т*кенами. Это токены предложение допустимые содержит, но недопустимую структуру все.

Когда вы читаете предложение на естественном языке или утверждение на формальном языке, вы должны понять структуру (хотя на естественном языке вы делаете это подсознательно). Этот процесс называется **синтаксическим разбором**, или **парсингом**.

Хотя у формальных и естественных языков много общих черт — токены, структура и синтаксис, — есть некоторые различия:

— *двусмысленность:*
  Естественные языки полны неоднозначности, с которой люди справляются с помощью контекстных подсказок и другой информации. Формальные языки спроектированы быть максимально однозначными, что означает, что любое утверждение имеет ровно одно значение независимо от контекста.
— *избыточность:*
  Чтобы компенсировать неоднозначность и уменьшить недопонимание, в естественных языках много избыточности. В результате они часто многословны. Формальные языки менее избыточны и более лаконичны.
— *буквальность:*
  Естественные языки полны идиом и метафор. Если я скажу: «Белая ворона», то, вероятно, я имею в виду не белую ворону или другую птицу, а «человека не такого, как все». Утверждения в формальных языках означают именно то, что они означают.

Поскольку все мы растем в среде естественного языка, иногда трудно приспособиться к формальным языкам. Различие между формальным

и естественным языком похоже на разницу между поэзией и прозой, более того:

— *Поэзия:*
  Слова используются как ради их звучания, так и ради их значения, и стихотворение в целом создает определенный эффект или вызывает эмоциональный отклик. Неоднозначность не только распространена, но и часто намеренна.
— *Проза:*
  Буквальное значение слов наиболее важно, а структура вносит больший смысл. Проза легче поддается анализу, чем поэзия, но все же часто неоднозначна.
— *Программы:*
  Значение компьютерной программы однозначно и буквально, и его можно полностью понять, проанализировав токены и структуру.

Формальные языки более насыщенные, чем естественные, поэтому их чтение занимает больше времени. Кроме того, структура важна, поэтому не всегда лучше читать сверху вниз, слева направо. Вместо этого научитесь анализировать программу в своей голове, выявляя токены и интерпретируя структуру. Наконец, детали имеют значение. Небольшие ошибки в написании и пунктуации, некритичные в естественных языках, важны в формальном языке.

ОТЛАДКА

Программисты делают ошибки. По интересной случайности* ошибки программирования называются багами (в пер. с англ. — жуками), а процесс их отслеживания называется **отладкой** (debugging).

Программирование и особенно отладка иногда вызывает сильные эмоции. Если вы долго боретесь с трудной ошибкой, то можете начать злиться или впадать в уныние.

Зачастую люди реагируют на компьютеры, как если бы те тоже были людьми. Когда они работают хорошо, мы считаем их коллегами, а когда они упрямы или грубы, мы реагируем на них соответствующим образом

* По самой распространенной версии, в 1946 году разработка компьютера Mark II была приостановлена из-за сбоя, которые был вызван попаданием мотылька между контактами (от англ. bug — жук, насекомое). *Прим. ред.*

(см. книгу Ривза и Насса, The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places).

Вот что поможет подготовиться к этим эмоциям. Один из подходов состоит в том, чтобы воспринимать компьютер как работника — со своими сильными сторонами, такими как скорость и точность, и с недостатками, такими как отсутствие сопереживания и неспособность понимать общую картину.

Ваша работа — стать хорошим управленцем: найти способы использовать сильные и слабые стороны. И найти способы использовать свои эмоции для решения проблемы, не позволяя им снижать эффективность работы.

Учиться отладке нелегко, но это ценный навык, полезный не только для программирования. В конце каждой главы есть раздел с моими предложениями по отладке. Надеюсь, что они помогут!

СЛОВАРЬ ТЕРМИНОВ

Решение задачи (problem solving):
Процесс формулирования задачи, поиска решения и его реализации.

Высокоуровневый язык (high-level language):
Язык программирования, разработанный, чтобы программистам было легко и удобно его использовать. Python — это высокоуровневый язык.

Низкоуровневый язык (low-level language):
Язык программирования, приближенный к инструкциям, которые понимает компьютер, но сложный для человека; также называется «машинным языком», или «языком ассемблера».

Портативность (portability):
Способность программы функционировать на компьютерах разного типа.

Интерпретатор (interpreter):
Программа, которая анализирует, обрабатывает, считывает и выполняет исходный код.

Приглашение (prompt):
Символы, отображаемые интерпретатором для обозначения ожидания ввода пользователем.

Программа (program):
Набор инструкций, определяющих вычисления.

Инструкция печати (print statement):
Инструкция, которая заставляет интерпретатор Python отображать значение на экране.

Оператор (operator):
Специальный символ (или символы), позволяющий выполнить простые вычисления, такие как сложение, умножение или соединение строк.

Значение (value):
Одна из основных единиц данных, таких как число или строка, которыми манипулирует программа.

Тип (type):
Тип — это категория значения. Основные типы переменных: целые числа (int), числа с плавающей точкой (float), строки (str).

Целое число (int):
Тип, который представляет целые числа.

Число с плавающей точкой (floating-point):
Тип, представляющий числа с дробной частью.

Строка (str):
Тип, который представляет собой последовательности символов.

Естественный язык (natural language):
Язык, на котором говорят люди и который развивался естественным путем.

Формальный язык (formal language):
Язык, который люди разработали для определенных целей, таких как представление математических концепций или компьютерных программ; все языки программирования — формальные языки.

Токен (token):
Один из основных элементов синтаксической структуры программы, аналог «слова» в естественном языке.

Синтаксис (syntax):
Правила, определяющие структуру исходного кода программы.

Парсинг (parsing):
Разбор кода программы и анализ синтаксической структуры.

Ошибка (bug):
Ошибка в программе.

Отладка (debugging):

Поиск и исправление ошибок.

УПРАЖНЕНИЯ

Упражнение 1.1

Рекомендуется прочитать эту книгу, сидя за компьютером, чтобы вы могли попробовать выполнить примеры самостоятельно по мере необходимости.

Каждый раз, когда экспериментируете с новой функцией, вы должны попытаться сделать ошибку. Например, что произойдет с программой "Hello, world!", если вы пропустите одну из кавычек? А если обе? А если вы напишете `print` с ошибкой?

Такие эксперименты помогут вам не только запомнить прочитанное, но и поспособствуют эффективному программированию, так как познакомят вас с основными сообщениями об ошибках.

Лучше ошибаться сейчас и нарочно, чем позже и случайно.

1. Что произойдет с инструкцией печати, если вы пропустите одну из скобок? Обе?
2. Тот же вопрос, но если вы пропустите одну из кавычек? Обе?
3. Вы можете использовать знак минус, чтобы указать отрицательное число, например `-2`. Что произойдет, если вы укажете знак плюс перед числом? К чему приведет код `2++2`?
4. В математике нули в начале — это абсолютно нормально, например, так: `02`. Что произойдет, если вы попробуете это сделать в Python?
5. Что произойдет, если указать два значения без оператора между ними?

Упражнение 1.2

Запустите интерпретатор Python и используйте его в качестве калькулятора.

1. Сколько секунд в 42 минутах и 42 секундах?
2. Сколько миль в 10 километрах? Подсказка: одна миля равна 1,61 км.
3. Если вы пробежали 10 километров за 42 минуты 42 секунды, каков ваш средний темп бега (время, затраченное на преодоление мили, в минутах и секундах)? Какова ваша средняя скорость в милях в час?

издательство
МАНН, ИВАНОВ И ФЕРБЕР
Максимально полезные книги